# Задание

Есть компания, у компании могут быть сотрудники. Сотрудник характеризуется именем, датой поступления на работу, базовой ставкой (для простоты, это значение по-умолчанию одинаково для всех видов сотрудников).  
Сотрудники бывают 3 видов - Employee, Manager, Sales. У каждого сотрудника может быть начальник. У каждого сторудника кроме Employee могут быть подчинённые.  
Зарплата сотрудника Employee - это базовая ставка плюс 3% за каждый год работы в компании, но не больше 30% суммарной надбавки  
Зарплата сотрудника Manager - это базовая ставка плюс 5% за каждый год работы в компании (но не больше 40% суммарной надбавки) плюс 0,5% зарплаты всех подчинённых первого уровня  
Зарплата сотрудника Sales - это базовая ставка плюс 1% за каждый год работы в компании (но не больше 35% суммарной надбавки) плюс 0,3% зарплаты всех подчинённых всех уровней  
У сотрудников (кроме Employee) может быть любое количество подчинённых любого вида.  
  
Требуется: составить архитектуру классов, описывающих данную модель, а также реализовать алгоритм расчета зарплаты каждого сотрудника на произвольный момент времени (а также подсчёт суммарной зарплаты всех сотрудников фирмы в целом) с помощью c# (веб сервис/консоль/пользовательский интерфейс на выбор, это не существенно для данной задачи).

# Decomposition

|  |  |
| --- | --- |
| R1 | Есть компания, у компании могут быть сотрудники.  (Создание репозитория с сотрудниками компании). |
| R2 | Есть класс сотрудников. Сотрудник характеризуется именем, датой поступления на работу, базовой ставкой.  (Создание класса сотрудников с необходимыми свойствами) |
| R3 | Сотрудники бывают 3 видов - Employee, Manager, Sales.  (Реализация отличий сущностей сотрудника) |
| R4 | Общее у всех сотрудников: есть начальник  (Добавление состояния наличия босса. Босс если есть подчиненные) |
| R5 | Наличие подчиненных у Manager |
| R6 | Наличие подчиненных у Sales |
| R7 | Расчет зп. Emploee: базовая ставка плюс 3% за год. Определить количество лет работы в компании. Убедиться, что не больше 30% суммарной надбавки |
| R8 | Расчет зп. Manager : базовая ставка плюс 5% за год. Определить количество лет работы в компании. Убедиться, что не больше 40% суммарной надбавки. Плюс 0,5% зарплаты всех подчинённых первого уровня ( Подзадача: найти всех подчиненных первого уровня) |
| R9 | Расчет зп. Sales - это базовая ставка плюс 1% за каждый год работы в компании (но не больше 35% суммарной надбавки) плюс 0,3% зарплаты всех подчинённых всех уровней (Подзадача: найти всех подчиненных всех уровней) |
| R10 | У сотрудников (кроме Employee) может быть любое количество подчинённых любого вида. |
| R11 | Необходим подсчёт суммарной зарплаты всех сотрудников фирмы в целом |
| R12 | Необходим расчет зарплаты каждого сотрудника на произвольный момент времени |

Требования

# English Description

После беглой декомпозиции этой большой задачи я выделил основные логические под задачи. Описал их в таблице выше. В задаче не указано как будет использоваться эта библиотека классов и поэтому не понятно как оптимальней построить модель данных. От модели данных зависит общая архитектура для этого решения.

На первый взгляд пришло 2 решения. Эти решения полностью исключают друга по своей архитектуре и идеологии. Заранее говорю, что я выбрал Решение 2.

## Inheritance-based architecture(not recommended)

У нас есть сущность с общей функциональностью – это Emploee. Другие сущности наследуются от этой базовой сущности. За счет наследования и полиморфизма будет достигаться отличие в поведении и характеристиках сущностей. То есть за счет полиморфизма будет выполняться подсчет зарплат по-разному. По заданию мы имеем неограниченный уровень вложенности. Ещё мы не знаем сколько на каждом уровне может быть сущностей.

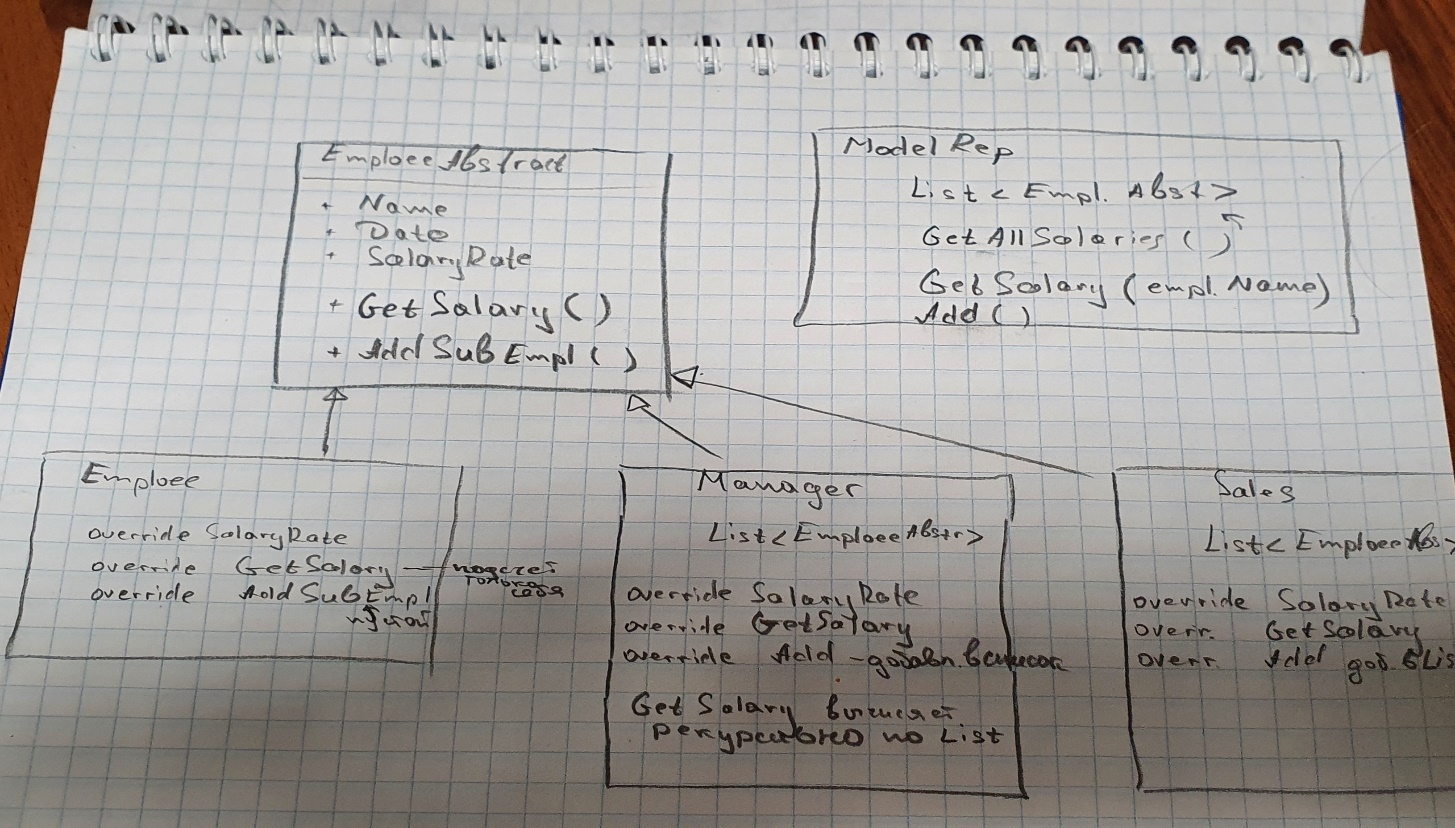
Поэтому одним из вариантов построения архитектуры – это использование шаблона GOF – Компоновщик для вычисления зарплаты каждого из сотрудников.

В этом случае мы будем работать с деревом обьектов как с одним объектом. Расчет зарплат будет инкапсулирован в каждой конкретной сущности и будет учитывать настройки и возможности этой сущности.

Необходим контейнер для хранения всех работников в классе – репозитории. Репозиторий будет иметь 2 метода:

* Посчитать зарплаты всех сотрудников. Сумма по коллекции сотрудников, вызывая для каждого метод GetSalary.
* Посчитать зарплату конкретного сотрудника. Вызов метода GetSalary для конкретного сотрудника из коллекции.

Простите за диаграмму в виде рисунка, не было достаточно времени).



Плюсы:

* Логическое разделение на сущности по признакам поведения.
* Разделение на слой – репозиторий и слой модели, состоящий из 3 sub entities.
* Изменение настроек сущности (базовый тариф) происходит в классе конкретной сущности

Минусы:

* Для сотрудников уровня Emploee некоторые методы будут пустыми в случае реализации паттерна Компоновщик
* Сложно адаптировать для сохранения в базу данных из-за разнообразия сущностей
* Для внесения изменений (базовый тариф) в трех сущностях – необходимо изменить 3 класса.
* Сложное восприятие общей архитектуры.
* Дублирование кода.
* Сложно поменять работнику свой тип. (например, трудно из Sales уйти в Manager)

Принял решение отказаться от этого варианта, потому что такого рода задачи часто нужно хранить в базе данных. И нужно более сильное разделение на зоны ответственности

## Independent Layers. Gof factory, strategy

Сущность Emploee универсальна. Отличия в поведении можно вынести в отдельный класс с общим интерфейсом и реализовать конкретные классы с отличным поведением.

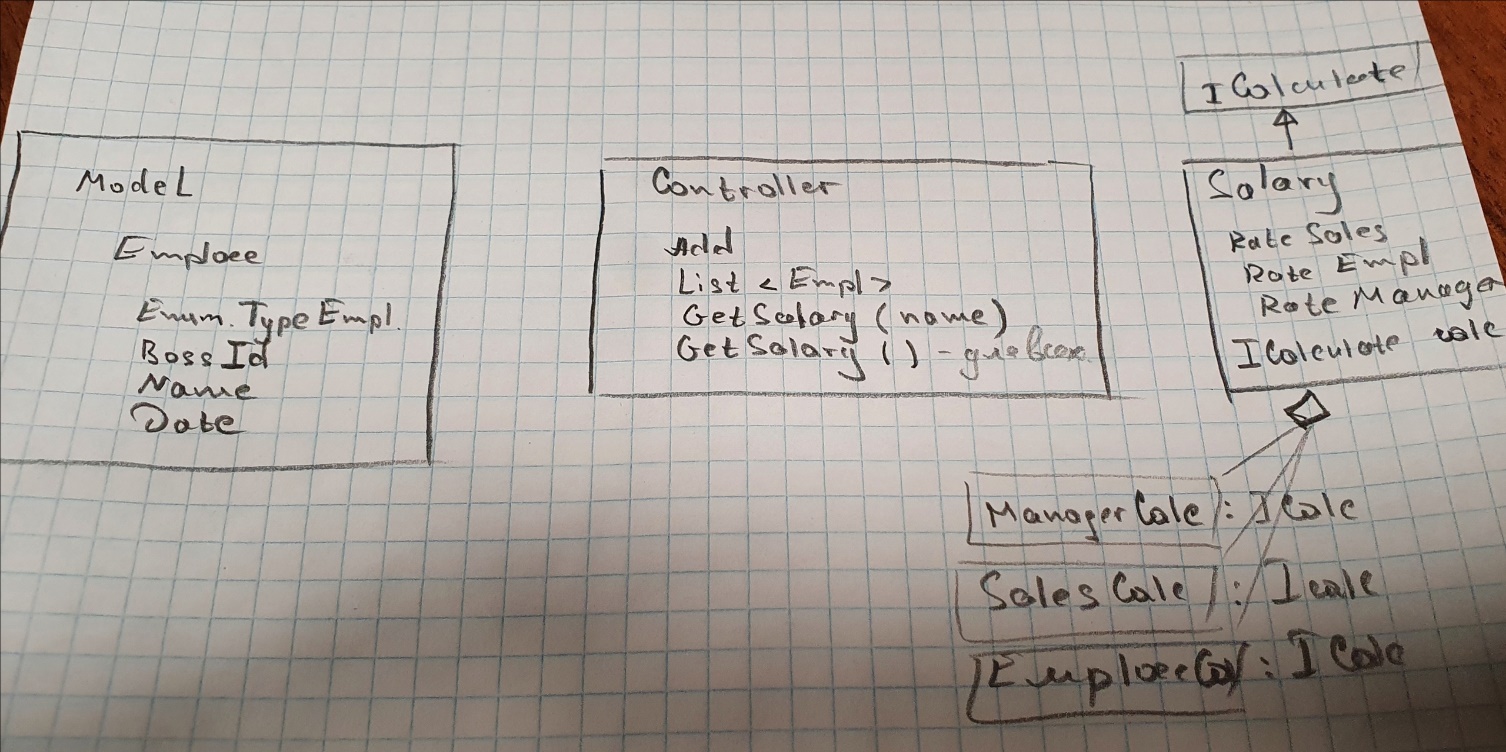
Предлагаю разделить на 3 независимых друг от друга слоя:

* Model Layer. Содержит только описание сущности Emploee. Это позволит легко импортировать данные в базу данных в будущем.
* Emploee Controller. Отвечает за создание работников, определяется тип работника и доступны ли ему подчиненные. Реализует паттерн GOF – фабричный метод для создания работников.
* DataRepository. Содержит коллекцию всех работников всех типов. Предоставляет методы добавления, удаления и доступа к колекции.
* Salary Layer. Отвечает за подсчет зарплаты. Реализует GOF паттерн – стратегия для подсчета зарплаты в зависимости от типа работника.

У каждого работника будет Enum с типом кем он является.

У каждого работника будет поле Id работника кто является его начальником.

Таким образом, у нас будет одна коллекция (в будущем база данных). Процесс подсчета зарплаты можно описать так: поиск всех работников удовлетворяющих критериям типа, вызов метода GetSalary(), который в зависимости от типа работника будет считать зарплату по разным алгоритмам ( Стратегия). Далее методы агрегации позволят получить суммы, если нужно.



Плюсы:

* Четкое разделение на слои по зонам ответственности
* Изменения по тарифам зарплаты – в одном месте в слое Salary
* Легкое понимание архитектуры
* Простая адаптация для базы данных
* Простой переход работника из одного типа в другой (например, из manager в sales)

Минусы:

* Необходимо перебирать всех работников и проверять их тип и Id начальника для операций подсчета зарплаты. Сложность O(n). Реализация создания Работников через GOF Фабрику кажется излишне сложной и не нужной для данной задачи. Фактически, сейчас отличие заключается в одном лишь типе. Но если в будущем надо будет добавить полезный метод отличающийся для разных типов работников то это будет очень легко сделать.

Что можно улучшить? Можно добавить базу данных. Для оптимизации поиска по базе можно добавить индексы по полю Тип работника, это ускорит поиск, сложность будет стремиться к O(log(n)). В базе данных можно хранить заранее просчитанные зарплаты каждого сотрудника. Но тут возникает задача актуализации данных в случае их изменений.